PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

OVERVIEW OF THE KNOWLEDGE BASE MANAGEMENT SYSTEM (KAPPA)

Kazumasa Yokota*, Moto Kawamura, Atsushi Kanaegami

Institute for New Generation Computer Technology (ICOT)
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan

ABSTRACT

This is an overview of the knowledge base management system called Kappa, one of the research and development activities performed at ICOT on databases and knowledge bases. The underlying data model is a nested relational model, and some complex data models are constructed and under consideration on the model. Deductive databases are constructed as one of knowledge base mechanisms and further extensions are planned within the framework of deductive and object-oriented databases. In this paper, we overview not only the current status of the system and the project but also their conception and activities related to them, and discuss the framework of deductive and object-oriented databases.

1 INTRODUCTION

Many knowledge information processing systems (KIPSs) have been developed and planned for the target of the Fifth Generation Computer System (FGCS) project. Most of them presuppose smaller or larger databases or knowledge bases, reflecting their own requirements, which are not usually based on traditional data models but on more complex and higher level data (knowledge) models. The term 'knowledge bases' generally means convenient black boxes with some kind of intelligence in each application, but that are located centrally. Unlike 'database', whose meaning is specific, 'knowledge base' is a very general term whose meaning ranges from naive databases to emulation of the human brain, and furthermore tends to depend on specific domains. At ICOT, under the various requirements of KIPSs and their environments, many kinds of research and development on database and knowledge base machines (DBMs and KBMs) and database and knowledge base management systems (DBMSs and KBMSs) have been done from different perspectives [Itoh86, Itoh+88] and will be integrated during the final stage of the FGCS project.

*e-mail: {enea,inria,kddlab,mit-eddie,ukc}!icot!kyokota; kyokota%icot.jp@relay.cs.net

Generally speaking, in our environment, an approach for knowledge-bases should be based on database, logic programming and artificial intelligence technologies, and should work towards their integration, regarding efficient performance, modeling power, clear and formal semantics, deductive mechanism, abstraction mechanism and parallel processing. In other words, the approach is a way of direct or declarative representation of objects in each domain of KIPSs and efficient processing of those objects on that level, without translation into the lower level language. As long as there is a problem of quantity, whether knowledge bases are stored in main memory or secondary memory, knowledge bases are not different from databases but are a developed or extended form of them.

The *Kappa* (Knowledge APPlication-oriented Advanced Database and Knowledge Base Management System) project started on September, 1985 as one of the KBMS projects at ICOT. The project was dedicated to the following environments: a personal sequential inference machine (PSI), its programming and operating system (SIMPOS), and a logic programming language (ESP) with 'object' concept; and multi-PSI and parallel inference machine (PIM) [Goto+88], its operating system (PIMOS) [Chik+88] and a parallel language (KL1) based on GHC. Under the first environment, the prototype system called Kappa-I has already worked not only as an experiment of research and development on KBMSs but also as tools for various applications.

This paper is an overview of the activities on KBMS, focusing on the Kappa project. Section 2 explains typical applications at ICOT, the basic policy for the KBMS, reflecting them, and the overall configuration. Section 3 describes some features of the database layer underlying the Kappa system, and briefly explains the architecture and some extensions under consideration. The knowledge base layer of Kappa is being constructed and designed on the database layer in the framework of deductive and object-oriented databases. Section 4 overviews the deductive features and Section 5 discusses the framework of deductive and object-oriented databases. Section 6 outlines the Kappa project and its related projects on the knowledge base systems at ICOT.

2 KNOWLEDGE BASES AT ICOT

Knowledge base facilities play an important role in many KIPSs, and are expected to make each knowledge base easy to create and utilize. From another point of view, it is very difficult to discriminate between data and knowledge, so free access is required for both data and knowledge from knowledge bases. For such purposes, Kappa project was planned to provide experimental environments of databases and knowledge bases for many applications. In that sense, the underlying frameworks or constraints are not independent of KIPSs developed or planned at ICOT, and are also not merely for basic research. In this section, we explain some requirements of KIPSs and the design principles of Kappa based on them.

2.1 Typical Applications

As Kappa presupposes conditions required by some applications, its universality, which reserves the independence of the applications' specific domains, depends on selection of applications from the viewpoints of importance, relevance, prospects and requisition of knowledge bases used by KIPSs. We analyzed many KIPSs in our environments and considered two applications, which are expected to be kernel systems in FGCS project, as typical and decided to address their requirements in the design of Kappa.

The first application is a proof checking system called *computer aided proof* (*CAP*) [Hirose+87, Sakai88], which requires various kinds of mathematical knowledge bases. The system has functions of theorem prover, term rewriting system and proof compiler (realizer), and is not only an artificial intelligence system but also expected to be one of the kernels of various problem solving systems. The system is based on Gentzen's natural deduction system (NK) with additional inference rules, and its proof description language (PDL) reflects the inference mechanism. Mathematical knowledge is classified into the form of texts (axioms, definitions, theorems and proofs) written in PDL, and the form of terms (proof trees during proof checking, and inference rules extracted from checked or assumed theorems). The unit of mathematical knowledge is a theory and the knowledge base constitutes a directed acyclic graph by reference relation between theories. There are various kinds of databases and knowledge bases in the system. In particular the inference mechanism, for a forward and backward reasoning mechanism during filling the gap between lines in a proof text, corresponds to query processing in deductive databases.

Another application is natural language processing systems, which play an important role in many KIPSs. These systems require many large *electronic dictionaries* as indispensable parts [Ishi+85, TaYo88]. These dictionaries are under construction and are a big step towards building huge knowledge bases. There are three kinds of dictionary:

- Electronic word dictionaries, each of which has a few hundred thousand words. Each dictionary constitutes a typical nested relation.
- A concept dictionary, which is a classification hierarchy of IS_A relations with several hundred thousand concepts.
- A thesaurus in the form of a semantic network, which is also an intermediate form between a word dictionary and a concept dictionary.

Furthermore, as many natural language processing systems are written in a logic programming language called CIL based on situation semantics [Mukai87], familiarity with CIL is also expected. In this domain, we face a problem of quantity as well as one of complex data structure.

2.2 Design Policy for the Kappa System

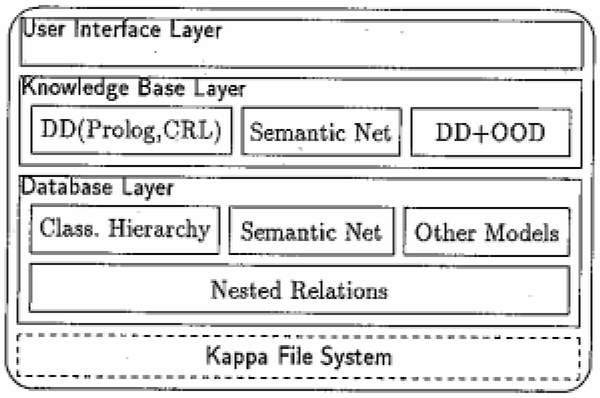
Considering the above requirements, a policy for the design of the Kappa system was set up:

- In our environment, knowledge bases should contain databases, and be considered as the database's extension.
- The system consists of three layers: *database*, *knowledge base* and *user interface* layers. Users can define each interface of the layers for their own applications and access any of them, that is, each layer should be extensible.
- The underlying data model of the database layer is a *nested relational model*, and some structured models such as *semantic network* and *classification hierarchy* are supported on nested relations in the layer. Terms expressing rules or structured data should be treated as a data type and be retrieved by unification and pattern matching.
- At least in the database layer, efficient performance for processing a large quantity of data is necessary to provide experimental environments for many KIPSs. Furthermore, as one of our environments consists of many personal workstations (PSIs) connected by a network, the system should be a distributed DBMS and KBMS, which features are different from some features of traditional centralized DBMSs.

- The knowledge base layer consists of some knowledge representation languages and various kinds of experimental modules based on them, such as deductive mechanism and object management in the framework of *deductive and object-oriented databases.*
- The user interface layer provides convenient interfaces and experimental features with graphic facilities, a structured editor (shared with CAP), and semi-automatic design tools for various kinds of structured data.
- For software development, we employ *object concept* in the physical and logical design, such as database object, relation object, tuple object and schema object. This is very different from conventional designs of DBMSs.

2.3 Configuration of the KBMS

According to the above policy, the first version of the prototype system (Kappa-I) was implemented in August 1987. The second version called Kappa-II is now being built and will be widely released next April (see Section 6, for details). Furthermore the Kappa system and other systems for DBMS and KBMS will be integrated into one system at the final stage of the FGCS project. The current overall configuration is shown in the following picture:



In the picture 'DD' and 'OOD' mean deductive databases and object-oriented databases respectively. In the following sections, we explain the database layer and deductive database in the knowledge base layer, and discuss the conception of deductive and object-oriented databases.

3 DATABASES FOR KNOWLEDGE BASES

The database layer is expected to manage various kinds of structured data and to process them efficiently. To cope with such requirements, we employ some data models in the layer. The underlying one is a nested relational model, and some other models are supported on the model.

3.1 Nested Relations

The advantages of a nested relational model over an ordinary relational model are that it offers more efficient representation and processing. In the ten years since the advantages were pointed out in [Maki77], there have been many works but few implementations [Verso86, Dadam+86, ScWe86, ScSc87, DeGu88]. It is also widely known that a nested relational model is better than a relational model for new applications such as engineering, office and geographical databases; and many commercial DBMSs also employ the idea from a practical point of view. However, as there are some variants of the name, the formal semantics should be made clear.

A nested tuple is defined as a subclass of objects, which notation is according to CRL [Yokota88]: assume a set O of atomic objects, a set A of attribute names, a tuple constructor [,], and a set constructor {, }. O may contain a special object ω to represent explicitly a void or null object, which is used for partial information of a tuple object. An *object* is defined as follows:

(1) Any atomic object is an object.

(2) If $o_1, \cdots, o_n$ are objects, then $\{o_1, \cdots, o_n\}$ is an object, which is called a *set object.*

(3) If $a_1, \cdots, a_n$ are different attribute names and $o_1, \cdots, o_n$ are objects, $[a_1/o_1, \cdots, a_n/o_n]$ is an object, which is called a *tuple object.* a_i/o_i is called a *tuple element.*

If a set object has only one element, the set brackets are omitted. A tuple object is called a *nested tuple,* if all elements of each set object contained in the tuple object are compatible mutually and the structure of attribute names of a tuple object constitutes a finite tree domain without duplication of attribute names, In the case, an atomic object is rather called an individual. A *nested relation* is a set object consisting of nested tuples, each of which has structure compatible with other tuples, and the *schema* is defined as a type (abstract structure) defined in Section 5.2. And a *nested relational database* is a tuple object consisting of pairs of a relation (attribute) name and a nested relation. It is easy to see that these definitions correspond with those of ordinary nested relations.

In this formulation, we can give two kinds of semantics to a set constructor, especially connected with row-nest and row-unnest operations. Assume there are two tuples: $[a/c_1, b/\{c_2, c_3\}]$ and $[a/c_1, b/\{c_3, c_4\}]$. There are two possible tuples after application of a row-nest operation to the given tuples:

$$[a/c_1, b/\{c_2, c_3, c_4\}], \quad \text{or}$$
$$[a/c_1, b/\{\{c_2, c_3\}, \{c_3, c_4\}\}].$$

Each tuple is resulted by a set union or a set-of (set grouping) operation respectively. While an (extended) NF^2 model [Dadam+86, ScWe86] employs the second semantics, Verso model [Verso86] employs the first. LDL [Beeri+87] is also considered as the second case. We take the first, and give the semantics of a nested tuple as a set of only column-nested tuples, which is independent of row-nest and row-unnest operation. Under the semantics, users do not need to be conscious of the structure when they query a database. For example, according to the semantics, all of the following relations

$$\begin{aligned} R_1:&\ \{[a/c_1, b/c_3], [a/c_2, b/c_3], [a/c_2, b/c_4]\},\\ R_2:&\ \{[a/\{c_1, c_2\}, b/c_3], [a/c_2, b/c_4]\}, \quad \text{and}\\ R_3:&\ \{[a/c_1, b/c_3], [a/c_2, b/\{c_3, c_4\}]\} \end{aligned}$$

have the same meaning, and $[a/\{c_1, c_2\}, b/c_3]$ is implied by each of them. A nested relational model based on such semantics is a natural extension of a relational model, and its characteristics are more efficient representation and more efficient processing performance. The extended relational algebra including nest and unnest operations is reconstructed according to the semantics.

Strictly speaking, such relations are classified into unnormalized relations and nested relations from a structural point of view, depending on whether the relations can be reversible by row-unnest and row-nest operations, and also classified into value-oriented relations and expression-oriented relations from an operational point of view, depending on whether sets have intrinsic meaning or not [Miura87]. Although in this context we do not discriminate the differences explicitly, the system supports the both. From an operational point of view, the extended algebra suppresses treatment of subtuples for selection, set operations, nest and unnest operations if the relations are value-oriented; the algebra manipulates subtuples according to the above semantics if the relations are expression-oriented. From a structural point of view, for nested relations in the strict sense, the system should support some automatic transformation from a given relation to the corresponding 'normal form' by using semantic constraints such as multi-value dependency; this is under consideration.

The actual model has some additional features for practical use: list and bag constructors, term as a data type and its retrieval by unification or pattern matching, and use of a constraint logic programming language CAL [Aiba+88] as generation or integrity rules for algebraic constraints. The extended relational algebra corresponds to such features, and furthermore supports some convenient operators. For example, let R be a nested relation, S be the schema and A be the subschema. In an intuitive notation of the algebra, the following operation

$$(\pi_{S\setminus A}\sigma_{Cond}(R)) \ltimes_{S\setminus A} R$$

is frequently used for some condition $Cond$ that includes a subset of A. The processing of this operation does not need to read tuples if $S \setminus A$ has a key property. Such an operation is supported as one algebraic operator for efficient processing.

Further extensions for nested relations are now under consideration: introduction of set grouping semantics and its corresponding operations, semi-automatic design support by using semantic constraints, parallel processing for Kappa-P (see Section 6), and some more built-in functions both for practical use and for efficient processing.

3.2 For More Structured Data

While a nested relational model is flexible and efficient for structured data used by many KIPSs, it fails to represent semantic relations between entities (objects or tuples). In practice, for the purpose, two data models are supported on nested relations. The first one is a *classification hierarchy* connected by a single link such as IS_A or HAS_A, and the second one is a *semantic network*. They are very useful for a concept dictionary and a thesaurus in natural language processing. This layer gives operations only for data manipulation (including simple inheritance) of nodes and links in hierarchies or networks, although such structures usually need more semantic manipulation along the links. More semantic operations will be supported on the upper knowledge base layer.

Other extensions on nested relations are being considered to provide an integrated environment, especially for a workstation environment: text as a data type and combination with interface facilities such as editors, and interfaces including a usual file interface. In such an environment, a DBMS should contain a usual file interface and provide a common interface to secondary memory. A (nested) term relation [Itoh86] is also considered using terms as a data type. Terms are convenient representations even with complex structure, but the semantics is outside the DBMS and the operations are expensive, so only the restricted form can be considered. In this layer, some more data models would be added, both to be practical and in order to support the above knowledge base layer.

3.3 Some Features of the DBMS

We overview some features of the system of the database layer. The prototype system called Kappa-I already works for various KIPSs and the second version called Kappa-II is under construction. In this subsection, we explain Kappa-II.

The database for nested relations resides in both main and secondary memories. Main memory is used as a cache and as a main memory database accompanying the delayed update process. For access to secondary memory, a new file system was implemented in order to make it possible to assign physical disk pages effectively and schedule physical accesses to them. Briefly, secondary memory for a database is divided physically into control, index, record, temporary, buffer-swap and log areas. The control area has information of database identity and the physical layout of the database. Each nested tuple is encoded into one string and stored in the record area. Data that relate to one relation are put as close together as possible in each of index and record areas. The temporary area is used for storing and manipulating intermediate relations.

One of the unique features of our system is a tuple identifier (TID), which identifies with one nested tuple. As the extended relational algebra can manipulate subtuples in the narrower sense of nested relations, according to the above semantics, a real TID consists of a set of sequences of the original TID and subTIDs identifying to subtuples. Indexes contain such sequences and support such mechanism. Intermediate relations as subsets of the original relation are in the form of a set of such TIDs. While unnest and nest operations are required during set operations to reserve the nest sequence of the schema, it is possible not by reading the corresponding values but by using only subTIDs. For example, assume the following relation with one nested tuple:

$$\{[a/k_1, b/\{d_1, d_2\}, c/\{d_3, d_4\}]\}.$$

For queries $b = d_1$ and $c = d_3$, the resulting relations are as follows:

$$\{[a/k_1, b/d_1, c/\{d_3, d_4\}]\}, \quad \text{and} \\ \{[a/k_1, b/\{d_1, d_2\}, c/d_3]\}.$$

If a union operation is applied to these relations, according to the above semantics, the result is

$$\{[a/k_1, b/d_1, c/\{d_3, d_4\}], [a/k_1, b/d_2, c/d_3]\}, \quad \text{or} \\ \{[a/k_1, b/\{d_1, d_2\}, c/d_3], [a/k_1, b/d_2, c/d_4]\},$$

where a/k_1 is not a tuple's key but a key of a set of tuples, whereas Verso generates the original relation to reserve the property of a tuple's key. Which relation should be taken depends on the nesting order between a and b, and the processing does not need to read tuples. Such use of identifiers is similar with ANDA [DeGu88], but the index structure is different: in our system, indexes are separate for each attribute, and each index entry contains a set of homogeneous sequences of a TID and subTIDs.

Familiar database facilities such as resource management, concurrency control and user management are of course equipped. For distributed facility, a server DBMS is assigned in each domain in PSI-network and controls all databases in the domain. Features of a distributed DBMS such as two phase commit and replication are not supported because they are not required so much in a personal workstation environment.

As the system is written in ESP, its object concept is used in the design and the development, and make the system simpler. Various logical elements such as a database, relations, sets, tuples, schemata and so on are all in the form of objects, and are thrown mutually between modules, which are also in the form of objects. An interface between user programs and the system is defined as methods of objects. There are two kinds of methods: system built-in methods such as extended relational algebra, and user-defined methods. Users can define any interface in the form of methods of an object for their applications and register the object to the system.

4 DEDUCTIVE DATABASES

The knowledge base layer consists of some subsystems such as deductive mechanism based on some languages, and inference mechanism of more structured data. Each deductive and inference mechanism is provided according to each knowledge representation language. We focus on the deductive database (DD) mechanism in this section and discuss structured data in next section.

A DD is an extension of a relational database in the light of logic. In other words a DD is the proof-theoretic reconstruction of a relational database [GMN84]. We consider DDs as a first step towards knowledge bases. As mentioned in Section 2, CAP system is one of the applications that needs deductive mechanism during proof checking. We have developed the prototype system based on definite clauses which correspond to relations without a set constructor as a proper subclass of nested relations, and we are now developing query processing based on CRL for a subclass of nested relations.

In the first step, we implemented query processing mechanism based on definite clauses: *sideways information passing* [Ullman85], *generalized magic sets* [BeRa87] and *semi-naive evaluation* [Ban86]. We plan to combine it with the CAP system. Some extensions are now under consideration: query optimization called Horn clauses transformation by restrictor (HCT/R) [Miya+88] in less restricted form than generalized magic sets, query processing by using semantic relations such as equivalence relation [Sakama88], query evaluation for stratified databases under the standard model semantics [Seki88], indefinite answer as a set of constraints combined with CAL [Aiba+88], and integrity constraint for update of DDs.

The next step aims at deductive mechanism on nested relations, for which we proposed a logic programming language called *CRL* [Yokota88]. It uses an attribute-valued notation instead of a predicate notation. Assume a set of variables besides the symbols of the nested tuples, the *term* is defined as follows:

(1) An atomic object o is a term,

(1') A variable X is a term,

(2) If $t_1, \cdots, t_n$ are terms, then $\{t_1, \cdots, t_n\}$ is a term, which is called a *set term*,

(3) If $a_1, \cdots, a_n$ are attribute names and $t_1, \cdots, t_n$ are terms, then $[a_1/t_1, \cdots, a_n/t_n]$ is a term, which is called a *tuple term*.

This attribute-valued notation is an extension of a usual predicate notation by introducing a set of some attribute names.:

$$p(t_1, \cdots, t_n) \Rightarrow [\$0/p, \$1/t_1, \cdots, \$n/t_n],$$

where $\$0, \$1, \cdots, \$n$ are newly introduced attribute names. The advantages are flexibility in the position and the number of arguments, and natural representation of column and row nesting tuples by set and tuple constructors. The same restrictions as nested tuples are also imposed on the terms for nested relations. Furthermore the notation can be used for more structured data (see Section 5.2).

For the semantics of the term, corresponding to the semantics of nested tuples, a term is mapped into a set of *partially tagged trees* (*PTTs*), each of which is defined as a partial function from a set of leaves of a tree domain of attribute names to a set of atomic objects without a void object: for a set M of PTTs, a variable assignment η and a tuple t,

$$[M;\eta] \models t \Leftrightarrow t\eta \in M.$$

While a PTT is given for the semantics of CIL [Mukai87], the term in this case is interpreted as a set of PTTs because of the set constructor. The semantics of the terms reserves the semantics of nested tuples. According to the semantics, row-nest and row-unnest operations correspond to a distributive law under an algebraic structure with tuple and set constructors:

$$\{[a/c_1, b/\{c_2, c_3\}]\} \Leftrightarrow \{[a/c_1, b/c_2], [a/c_1, b/c_3]\}.$$

Hence the syntax of terms is restricted to a multi-valued level so that unification can be made decidable and a logic programming language constructed, although the database with only ground terms (nested tuples) may have more deeply nested tuples. Unification between the terms are defined by merge (overlapping) of tree domains of attribute names, set intersection between subterms of set type, which reduced to non-empty set, and atomic object identity.

A *program clause* is a pair of a tuple term t and a set $\{t_1, \cdots, t_n\}$ of tuple terms. A pair $(t, \{t_1, \cdots, t_n\})$ is written as follows:

$$t \leftarrow t_1, \cdots, t_n.$$

A *CRL program* is a set of clauses. And a *goal* is a set of tuple terms $q_1, \cdots, q_m$, which is written as $\leftarrow q_1, \cdots, q_m$, and a *database query* is a query with one term, which does not mean loss of generality. For example,

$$\begin{array}{l} [par/\{``mary"\}, chi/\{``john", ``lisa"\}], \\ [par/\{``paul", ``kate"\}, chi/\{``mary"\}], \\ [anc/X, des/Y] \leftarrow [par/X, chi/Y], \\ [anc/X, des/Y] \leftarrow [par/X, chi/Z], [anc/Z, des/Y] \end{array}$$

is a CRL program. A *CRL database* is a CRL program and is divided into an intensional and an extensional databases (an IDB and an EDB), just like a definite clause based DD. An EDB is a set of nested tuples.

For a program or a database P, a non-empty set M of PTTs, which is constructed by PTTs included in P, is called a *model* of P when M satisfies the following condition:

$$\forall q \leftarrow p_1, \cdots, p_n \in P(M \models p_1 \wedge \cdots \wedge p_n \supset M \models q).$$

The definition shows the following properties:

$$\begin{array}{l} q \leftarrow p_1, \cdots, p_n \\ \equiv q \leftarrow p_1, \cdots, p_{i-1}, p_{i1}, \cdots, p_{im}, p_{i+1}, \cdots, p_n \\ \equiv q_1 \leftarrow p_1, \cdots, p_n \wedge \cdots \wedge q_k \leftarrow p_1, \cdots, p_n, \end{array}$$

where $p_{i1}, \cdots, p_{im}$ and $q_1, \cdots, q_k$ are 'unnested' results of p_i and q, respectively. The relation guarantees to reserve the declarative and procedural semantics of Prolog. As the extended SLD resolution for the CRL program, set unification (intersection) is applied for an EDB term without being unnesting, and the new goal corresponding to the set difference is generated for an IDB. For example, consider the following goal:

$$\leftarrow G_1, [\cdots, a/S, \cdots], G_2,$$

where S is a set term. If the subgoal $[\cdots, a/S, \cdots]$ can be unified with an EDB term $[\cdots, a/S', \cdots]$ by a unifier θ, the new goal is generated as follows:

$$\leftarrow (G_1, [\cdots, a/(S \setminus S'), \cdots], G_2)\theta,$$

if $S \setminus S'$ is not an empty set.

As for bottom-up evaluation, the least fixpoint semantics like Prolog is also defined, and similar optimization strategies can be applied. For example, for a query $\leftarrow [anc/\{``paul", ``kate"\}, des/X]$ ($\leftarrow [anc/``paul", des/X], [anc/``kate", des/X]$), the above

CRL database (program) can be transformed into the following form:

$$\begin{array}{l}
[par/\{\text{"}mary\text{"}\}, chi/\{\text{"}john\text{"}, \text{"}lisa\text{"}\}]. \\
[par/\{\text{"}paul\text{"}, \text{"}kate\text{"}\}, chi/\{\text{"}mary\text{"}\}]. \\
[anc/X, des/Y] \leftarrow [anc^*/X], [par/X, chi/Y]. \\
[anc/X, des/Y] \leftarrow [anc^*/X], [par/X, chi/Z], \\
\qquad\qquad\qquad\qquad [anc/Z, des/Y]. \\
[anc^*/\text{"}paul\text{"}]. \\
[anc^*/\text{"}kate\text{"}]. \\
[anc^*/Z] \leftarrow [anc^*/X], [par/X, chi/Z].
\end{array}$$

This is an example of HCT/R [Miya+88] for a CRL database.

Current CRL (intensional) databases are restricted to multi-valued nested relations, although the database layer supports unlimited nesting logically. Under the same semantics, looser restrictions are required. Some extensions are considered in addition to the case of definite clauses. The first extension is the introduction of set grouping semantics like LDL [Beeri+87], which corresponds to an extension of the database layer. Another extension is the creation of an attribute system that can compose attribute names, introduced in [Tanaka87], where an initial set of attributes, simply called a vocabulary dictionary, is extended by repeating both composition of attribute names and labeling the result. The introduction of such a dictionary also makes it possible to introduce 'constraints' by attribute composition.

5 TOWARDS DEDUCTIVE AND OBJECT-ORIENTED DATABASES

5.1 Deductive vs. Object-Oriented

In order to cope with wider applications including KIPSs, more complex data models which extend from flat relations to (nearly) direct representation of objects of the real world are required and proposed [BaKh85, Maier86, AbGr88, Lecluse+88, Beeri+88, Beeri88]. These data models includes various elements: data modeling in the database area; data abstraction, type disciplines and object concept in programming language; and type inheritance in knowledge representation language. We set up the direction of knowledge bases in our environment as representation of 'objects' and their processing in the framework of DDs, (as integration of DDs and object-oriented databases (OODs)).

The formalization of nested relations as one of 'objects' is tried or pointed out in various contexts: type inheritance [Ait86], complex objects [BaKh85, AbGr88], objects [Maier86] and logic programming with sets [Beeri+87]. These ways show a framework of formulation common to that of more structured data such as nested relations, complex objects and objects, although each specific object has its intrinsic operations, such as row-nest and row-unnest operations in nested relations. Such formulation is also related to other areas such as representation of situation in natural language processing [Mukai87]. We generally call such structured data 'objects', which include nested relations and complex objects.

OODs are proposed by stimulation of the success of object-oriented programming languages (OOPs). They are more appropriate for representation of 'objects' of the real world and have many possibilities such as providing the framework of multi-media databases, serving as dissolution of 'impedance mismatch', (filling gaps in data structure and data operation between database and programming languages), or helping to create classification hierarchy. However most of the approaches are rather practical and is not based on logic or mathematics. The concept is still vague and there is no consensus [Ban88].

An OOP is not the same as an OOD even though it would support data persistence, because OODs should support intrinsic objects in a database area, including nested relations and complex objects, and have traditional features of DBMSs as a management system. The term 'object-oriented' has two kinds of meanings: the structural aspect (static objects) such as complex structure, object identity, data abstraction and classification hierarchy; and behavioral aspects (active objects) such as methods, message passing and information encapsulation. OODs except Smalltalk-based databases focus mainly on the first aspect, whereas OOPs focus on the second aspect.

In a database area, an approach for DDs gives logical foundations and perspectives of extensions towards knowledge bases, whereas an approach for OODs gives the framework of 'object' modeling of the real world. Most of advanced applications including KIPSs require new databases (or knowledge bases) which support both powerful inference mechanism and high-level modeling capability. If OODs could be treated based on such formulation as an approach for DDs has proposes, they would serve many KIPSs even without the behavioral aspect, and more with the aspect.

For such purposes, many approaches for integration of DDs and OODs, i.e., a framework called *deductive and object-oriented databases* (*DOODs*) can be considered. We set up a DOOD as a framework of knowledge bases in our target. We started to take such an approach for a subclass of 'objects' (nested relations), which is also appropriate for many applications in our environments, and being developed a deductive mechanism based on CRL as a first step.

5.2 The Framework of DOOD

As mentioned above, the predicate notation like Prolog lacks flexibility for more structured data. Although it is considered to embed new expression into the predicate notation, it is only convenient but not essential; there seems to be no reason to persist in it. We use objects defined in Section 3.1. Such an object might be called a nested tuple or a complex object with some restrictions according to each domain.

In order to classify objects, a *type* is defined as follows:

(1) A void object ω belongs to a type $\top$ and all other atomic objects belong to a same type atom,

(2) If $o_1, \cdots, o_n$ belong to types, $\tau_1, \cdots, \tau_n$ respectively, then a set object $o = \{o_1, \cdots, o_n\}$ belongs to a set type $\{\tau_1, \cdots, \tau_n\}$.

(3) If each of $o_1, \cdots, o_n$ belongs to a type, $\tau_1, \cdots, \tau_n$, then a tuple object $o = [a_1/o_1, \cdots, a_n/o_n]$ belongs to a tuple type $[a_1/\tau_1, \cdots, a_n/\tau_n]$.

Note that atomic objects except ω do not necessarily belong to one type, and might be divided into some atomic types such as string and integer.

We add another type $\perp$ to a set of types and define a partial order $\preceq$ between types as follows:

(1) For any type τ, $\perp \preceq \tau$ and $\tau \preceq \top$.

(2) For set types $\tau_1 = \{\tau_{11}, \cdots, \tau_{1n}\}$ and $\tau_2 = \{\tau_{21}, \cdots, \tau_{2m}\}$, if $\forall \tau_{1i}, \exists \tau_{2j}(\tau_{1i} \preceq \tau_{2j})$, then $\tau_1 \preceq \tau_2$.

(3) For a tuple type $\tau = [a_1/\tau_1, \cdots, a_n/\tau_n]$, let τ' be a type, obtained by τ, which excludes a_i/τ_i such that $\tau_i = \top$. For tuple types τ_1 and τ_2, let $\tau_1' = [a_{11}/\tau_{11}, \cdots, a_{1n}/\tau_{1n}]$ and $\tau_2' = [a_{21}/\tau_{21}, \cdots, a_{2m}/\tau_{2m}]$. If there exists i such that $\tau_{1i} = \perp$, or $\{a_{11}, \cdots, a_{1n}\} \supseteq \{a_{21}, \cdots, a_{2m}\}$ and $\forall j, \exists i(a_{1i} = a_{2j} \wedge \tau_{1i} \preceq \tau_{2j})$, then $\tau_1 \preceq \tau_2$.

According to the definition, we can obtain the following equivalence relation $\approx$:

$$\begin{array}{rcl}
\{\tau_1, \tau_2, \cdots, \tau_n\} & \approx & \{\tau_2, \cdots, \tau_n\}, \text{ if } \tau_1 \preceq \tau_2, \\
\{\top, \tau_2, \cdots, \tau_n\} & \approx & \{\top\}, \\
\{\perp, \tau_2, \cdots, \tau_n\} & \approx & \{\tau_2, \cdots, \tau_n\}, \\
[a_1/\top, a_2/\tau_2, \cdots, a_n/\tau_n] & \approx & [a_2/\tau_2, \cdots, a_n/\tau_n], \\
[a_1/\perp, a_2/\tau_2, \cdots, a_n/\tau_n] & \approx & [a_i/\perp], \text{ for any } i.
\end{array}$$

Note that elements in a set object or a tuple object are commutative. Without loss of generality, we consider an equivalence class modulo $\approx$ and assume that each tuple element of a tuple object does not contain $\top$ or $\perp$, and each element of a set object is not ordered with others. A type τ_1 is *compatible* with a type τ_2 if there exists a type τ such that $\tau \neq \top$, $\tau_1 \preceq \tau$ and $\tau_2 \preceq \tau$. According to the definition, we exclude incompatible (heterogeneous) set objects, some elements of which belong to incompatible types. A type might be called a schema for nested relations. A set of types including $\top$ and $\perp$ constitutes a lattice, where g.l.b. and l.u.b. between set types correspond to set intersection and union respectively, and g.l.b. and l.u.b. between a set type and a tuple type result in $\perp$ and $\top$ respectively.

We can redefine objects by combining objects and types:

(1) For a null object ω, $\omega : \top$ is an object.

(2) For any atomic object o except ω, $o :$ atom is an object.

(3) If $o_1 : \tau_1, \cdots, o_n : \tau_n$ are objects and $\tau_1, \cdots, \tau_n$ are compatible, then $\{o_1 : \tau_1, \cdots, o_n : \tau_n\} : \{\tau_1, \cdots, \tau_n\}$ is a set object.

(4) If $a_1, \cdots, a_n$ are different attribute names and $o_1 : \tau_1, \cdots, o_n : \tau_n$ are objects, $[a_1/o_1 : \tau_1, \cdots, a_n/o_n : \tau_n] : [a_1/\tau_1, \cdots, a_n/\tau_n]$ is a tuple object.

(5) If $o : \tau_1$ is an object and $\tau_1 \preceq \tau_2$, then $o : \tau_2$ is an object.

Even if types are not attached explicitly, they can be inferred from the innermost.

New type symbols are introduced to define structured types and reserve the original ordering. A set type τ_s or a tuple type τ_t is can be redefined as follows:

$$\tau = \tau_s, \quad \text{or} \quad \tau = \tau_t,$$

where τ is a newly introduced type name. Furthermore, a type can be defined recursively and represent infinite structure, like tags in [Ait86]. For example,

$$\tau = [id/\text{integer}, name/\text{string}, age/\text{integer}, \cdots, des/\{\tau\}].$$

Corresponding to the type, an object can be represented by introducing object identities. For example,

$$\begin{array}{l}
[id/10, \\
name/\text{“}taro\text{”}, \\
age/30, \cdots, \\
des/\{[id/20], [id/30]\} : \{\tau\}] : \tau,
\end{array}$$

where obvious types are omitted and 'id' is an attribute name of an object identity.

In such formulation, various structured data such nested relations, complex objects and classification hierarchy can be developed. For the purpose, specific operations such as row-nest and row-unnest for nested relations, or specific orderings such as HAS_A relation in complex objects [BaKh85] and IS_A relation for classification hierarchy or nested relations with inheritance [Nakano88] should be introduced. We can consider such formulation as a general framework of DOOD, which is independent of specific domains, and construct various structured objects as an instance of the framework by introducing each intrinsic meaning and specific restrictions.

Some of them can be resolved into conjunction of simple forms, pointed out in [Beeri88]:

$$o = [a_1/o_1, \cdots, a_n/o_n] \Leftrightarrow o.a_1 = o_1 \wedge \cdots \wedge o.a_n = o_n.$$

Such translation makes the semantics clear but goes far from our intention such as nearly direct representation of 'objects' of the real world and its inference at the level.

In the above construction, further considerations remain:

- Mathematical structure for a set of objects with each specific domain, and the formal semantics. If their objects constitute a lattice such as [Ait86, BaKh85, Nakano88], the treatment would become easier.
- Restrictions to make up a logic programming language or an IDB in a DD based on such objects, for example, limitation for decidability of unification. Not only a fixpoint semantics such as [Beeri+87, AbGr88] but also a procedural semantics should be given.
- How to relate such objects to more general knowledge representation such as semantic networks and frames. Although knowledge is represented in the form of a rule from a viewpoint of DDs based on definite clauses, it might be represented in another form such as a set of orderings in DOOD. This is also a problem for integration of Kappa and ETA.
- An object itself and a relation between objects can be considered as constraints, and a logic programming language based on them can be within the framework of a constraint logic programming scheme $CLP(X)$ [JaLa87]. If we could consider the scheme of DOOD such as the above formulation, each specific meaning and operation could be given as a constraint solver in each domain.
- Even in the structural aspects of objects, an update problem [Maier86, Beeri88] should be solved for a 'real' extension of traditional databases.

Although we have not discussed the behavioral aspects of objects, there are already some researches such as [Beeri+88, Lecluse+88]. Such aspects can be considered from two viewpoints: uniform framework of interface including dissolution of impedance mismatch; data manipulation depending on kinds of objects such as text, picture or geographical data. There remain problems about how such an aspect fits a framework of DDs or OOPs. This approach also will contribute to many new applications.

6 OVERVIEW OF THE PROJECTS

We have overviewed the knowledge base management system, Kappa, and in this section explain the outline of the project and the other related projects: *ETA* and *PHI* projects. Kappa project started in September, 1985, and is divided into two-year subprojects.

The first subproject called, *Kappa-I*, started in September, 1985 and ended in August, 1987. It intended to make a DBMS that would reflect some requirements of CAP and electronic language dictionaries in its design, provide experimental environments for other KIPSs, and form the grounds for knowledge bases. The underlying data model of the DBMS is a nested relational model in the above sense, and has some advanced features such as terms as a data type. Classification hierarchy is also supported on nested relations. The size of the system consists of about 60,000 lines in ESP. The evaluation test shows that the system works efficiently, and some KIPSs already use the system as the underlying database.

The second subproject called, *Kappa-II*, started in April, 1987, overlapping with Kappa-I, and will end in March, 1989. The targets of the system are further improvement of Kappa-I, especially in processing performance and in management of main memory, introduction of semantic network as one of the basic objects, implementation of various interfaces, and implementation of the prototype of DDs both in definite clauses and CRL. We are now constructing the system; it will be released next April for users who engage in knowledge information processing systems on PSI.

The third subproject, *Kappa-III*, and the fourth subproject, *Kappa-P*, will start in December, 1988, also overlapping with Kappa-II. In Kappa-III, object flavors are added to all layers of Kappa-II, as mentioned in Section 5. More structured data than nested relations are supported, and knowledge bases are considered in the framework of deductive and object-oriented databases. Kappa-P is intended to do research and development on parallel processing of Kappa-II and providing experimental environments of databases and knowledge bases on Multi-PSI and PIM. These subprojects are mutually related: some parallel algorithms devised in the former will be also implemented in the latter.

The ETA is a knowledge object management system, devoted to knowledge representation in the form of semantic network and its intelligence information retrieval mechanism based on abstractive layers of the objects [Koguchi+88]. The prototype system was implemented on PSI, and shows such an approach to be very useful for structured knowledge such as text data. The language for structured data is being refined towards having clear semantics, and realizing deductive and inference mechanisms. The system is being designed to work on Kappa system.

The PHI project is intended to investigate the mechanism of a distributed DD system based on a relational database in PSIs and PSI-network environment [Itoh+88, Miya+88]. The prototype system was implemented on PSI, focusing on the superimposed code scheme for term access, recursive query optimization (HCT), and its distributed processing. Query optimization of stratified databases is under further consideration.

At the final stage of the FGCS project, these systems and various ideas for knowledge base management systems will be integrated, and will provide an experimental environment for many knowledge information processing systems.

7 CONCLUSIONS

Kappa is a software project among the knowledge base projects in ICOT. As the middle-range target of the knowledge base management system, we set up a framework of deductive and object-oriented databases. Although the two approaches for deductive databases and object-oriented databases have been taken almost independently, some work towards their integration has begun recently, and we are working towards integration. Among them the characteristics of our research and development can be summarized as follows:

- Knowledge bases in our environment are taken as an extended form of databases, especially for a deductive approach to structured data including terms, in the framework of deductive and object-oriented databases.
- The underlying data model of our system is a nested relational model because of the efficient internal representation and the efficient processing of structured data; more complex data models are constructed on the nested relational model.
- A deductive approach is taken first for nested relations as a subclass of 'objects', and being further considered for more structured data such as classification hierarchy, complex objects and semantic network.
- Some projects for databases and knowledge bases, such as Kappa, ETA and PHI are in cooperation with each other, and will be integrated with other systems such as PIM, PIMOS and KBM at the final stage of the FGCS project for the target.

Acknowledgments

The research and development described in this article is being done mainly by the members of the KBMS groups both in ICOT and the participating companies. The authors are grateful to Dr. Kazuhiro Fuchi and Dr. Shunichi Uchida for encouraging the projects and providing useful suggestion.

References

[AbGr88] S. Abiteboul and S. Grumbach, "COL: A Logic-Based language for Complex Objects", *EDBT*, in *LNCS*, 303, Springer, 1988

[Aiba+88] Z. Aiba, K. Sakai, Y. Sato, D. Hawley and R. Hasegawa, "Constraint Logic Programming Language CAL", *FGCS*, 1988

[Ait86] H. Ait-Kaci, "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *TCS*, vol.45, 1986

[BaKh85] F. Bancilhon and S. Khoshahian, "A Calculus for Complex Objects", *ACM PODS*, 1985

[Ban86] F. Bancilhon, "Naive Evaluation of Recursively Defined Relations", in *On Knowledge Base Management Systems*, M.L. Brodie, et al, eds., Springer, 1986

[Ban88] F. Bancilhon, "Object-Oriented Database Systems", *ACM PODS*, 1988

[Beeri88] C. Beeri, "Data Models and Languages for Databases", *ICDT*, 1988

[Beeri+87] C. Beeri, S. Naqvi, O. Shnueli and S. Tsur, "Sets and Negation in a Logic Database Language (LDL)", *ACM PODS*, 1987

[Beeri+88] C. Beeri, R. Nasr and S. Tsur, "Embedding ψ-term in a Horn-clause Logic", in *Proc. of Third Int'l Conf. on Data and Knowledge Bases*, Jersalem, 1988

[BeRa87] C. Beeri and R. Ramakrishnam, "On the Power of Magic", *ACM PODS*, 1987

[Chik+88] T. Chikayama, H. Sato and T. Miyazaki, "Overview of the Parallel Inference Machine Operating System (PIMOS)", *FGCS*, 1988

[Dadam+86] P. Dadam, et al, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", *ACM SIGMOD*, 1986

[DeGu88] A. Deshpande and D. Van Gucht, "An Implementation for Nested Relational Databases", *VLDB*, 1988

[GMN84] H. Gallaire, J. Minker and L.-M. Nicolas, "Logic and Databases: A Deductive Approach", *ACM Computing Surveys*, vol.16, no.2, 1984

[Goto$^+$88] A. Goto, M. Sato, K. Nakajima, K. Taki and A. Matsumoto, "Overview of the Parallel Inference Machine Architecture (PIM)", *FGCS*, 1988

[Hirose$^+$87] K. Hirose, K. Yokota and K. Sakai, "An Approach to Proof Checker", *ICOT-TR*, 224, 1987

[Ishi$^+$85] T. Ishikawa, H. Tanaka, et al, "Basic Specifications of the Machine-Readable Dictionary", *ICOT-TR*, 100, 1985

[Itoh86] H. Itoh, "Research and Development on Knowledge Base Systems at ICOT", *VLDB*, 1986

[Itoh$^+$88] H. Itoh, H. Monoi et al, "Outline of the Knowledge Base Subsystem", *FGCS*, 1988

[JaLa87] J. Jaffer and J.-L Lassez, "Constraint Logic Programming", *IEEE SLP*, 1987

[Koguchi$^+$88] T. Koguchi, H. Kondo, M. Oba and H. Itoh, "Knowledge Representation with Abstractive Layers for Information Retrieval", *FGCS*, 1988

[Lecluse$^+$88] C. Lecluse, P. Richard and F. Velez, "O_2, an Object-Oriented Data Model", *EDBT*, in *LNCS*, 303, Springer, 1988

[Maier86] D. Maier, "A Logic for Objects", in *Preprint of the Workshop on Foundations of Deductive Databases and Logic Programming*, 1986

[Maki77] A. Makinouchi, "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", *VLDB*, 1977

[Miura87] T. Miura, "Theory of Non First Normal Form Relational Databases – A Survey", in *Proc. of Advanced Database Symposium*, IPSJ, Tokyo, Dec., 1987 (in Japanese)

[Miya$^+$88] N. Miyazaki, K. Yokota, H. Haniuda and H. Itoh, "Horn Clause Transformation by Restrictor in Deductive Databases", *ICOT-TR*, 407, 1988

[Mukai87] K. Mukai, "Anadic Tuples in Prolog", *ICOT-TR*, 239, 1987

[Nakano88] R. Nakano, "Frame Lattice Model", in *Special Interest Group Notes of IPSJ*, Sep., 1988 (in Japanese)

[Sakai88] K. Sakai, "Towards Mechanization of Mathematics – Proof Checker and Term Rewriting System", in *Programming of Future Generation Computers*, K. Fuchi and M. Nivat, eds., Elsevier, 1988

[Sakama88] C. Sakama and H. Itoh, "Handling Knowledge by its Representative", *EDS*, 1988

[ScSc87] M.H. Scholl and H.-J. Schek, (eds.), *Theory and Applications of Nested Relations and Complex Objects – An International Workshop, Workshop Material*, 1987

[ScWe86] H.-J. Schek and G. Weikum, "DASDBS: Concepts and Architecture of a Database System for Advanced Applications", *Tech. Univ. of Darmstadt, TR*, DVSI-1986-T1, 1986

[Seki88] H. Seki and H. Itoh, "A Query Evaluation Method for Stratified Programs under the Extended CWA", *LP*, 1988

[Tanaka87] Y. Tanaka, "Roles of a Vocabulary in Knowledge-Based Systems", *IFIP WG 10.1 Workshop*, Gotenba, 1987

[TaYo88] Y. Tanaka and Y. Yoshioka, "Overview of the Dictionary and Lexical Knowledge Base Research", *FGCS*, 1988

[Ullman85] J.D. Ullman, "Implementation of Logical Query languages for Databases", *ACM TODS*, vol.10, no.3, 1985

[Verso86] J. Verso, "VERSO: A Data Base Machine Based on Non 1NF Relations", *INRIA-TR*, 523, 1986

[Yokota88] K. Yokota, "Deductive Approach for Nested Relations", *ICOT-TR*, 1988